What is Claimed is:

1. A computer-implemented method for increasing the execution performance of a function at run-time, the computer-implemented method comprising:

compiling the function;

identifying a call to a process, the call to the process being included in the function; and

adding dependency information to the function, wherein the dependency information is arranged to indicate a status of the function, the dependency information including class information, name information, and signature information associated with the process, wherein the status of the function is arranged to indicate a validity of the function and a compilation status of the function.

2. A computer-implemented method as recited in claim 1 wherein the process is a virtual process, the computer-implemented method further including:

analyzing a class structure associated with the function, wherein analyzing the class structure includes determining when the virtual process is a substantially unique target of the call.

3. A computer-implemented method as recited in claim 2, the computer-implemented method further including:

inlining the virtual process into the function when it is determined that the virtual process is the substantially unique target of the call.

4. A computer-implemented method as recited in claim 2, the computer-implemented method further including:

placing a direct call to the virtual process in the function.

5. A computer-implemented method as recited in claim 1 further including:

determining when the function is suitable for compilation.

6. A computer-implemented method as recited in claim 1 further including:

loading a class, the class being associated with the function; and

determining when the dependency information indicates that the function is valid, that the function is suitable for deoptimization, and that the function is suitable for reoptimization.

7. A computer-implemented method as recited in claim 6 wherein loading the class includes:

determining when the function is not a substantially unique caller to the process; and

de-compiling the function when is determined that the function is not a substantially unique caller to the process.

8. A computer-implemented method as recited in claim 6 wherein loading the class includes:

determining when the function is not a substantially unique caller to the process; and

re-compiling the function when is determined that the function is not a substantially unique caller to the process.

9. A computer-implemented method for analyzing a first class associated with a class hierarchy of a system during run-time, the computer-implemented method comprising:

marking the first class;

marking a second class, the second class being included in the class hierarchy, the second class further being associated with the first class, wherein marking the second class substantially identifies the second class as being associated with the first class;

inspecting a compiled function associated with the system, the compiled function including dependency information, the dependency information being arranged to indicate a validity status of the compiled function and the optimization status of the compiled function, wherein inspecting the compiled function includes determining when at least one of the first class and the second class is identified in the dependency information; and

determining when the compiled function is invalid when it is determined that at least one of the first class and the second class is identified in the dependency information.

10. A computer-implemented method as recited in claim 9 further including:

de-compiling the compiled function when it is determined that the compiled function is invalid, wherein de-compiling the compiled function effectively places the compiled function in an interpreted form.

11. A computer-implemented method as recited in claim 9 further including:

re-compiling the compiled function when it is determined that the compiled function is invalid, wherein re-compiling the compiled function allows the compiled function to account for the first class.

12. A computing system suitable for increasing the execution performance of a function at run-time, the computing system comprising:

a processor;

a compiler arranged to compile the function;

a call identifier arranged to identify a call to a process, the call to the process being included in the function; and

a mechanism suitable for adding dependency information to the function, wherein the dependency information is arranged to indicate a status of the function, the status being arranged to include when a validity status of the function.

13. A computing system as recited in claim 12 wherein the process is a virtual process, the computing system further including:

an analyzer for analyzing a class structure associated with the function, wherein analyzing the class structure includes determining when the virtual process is a substantially unique target of the call.

14. A computing system as recited in claim 13, the computing system further including:

an inliner arranged for inlining the virtual process into the function when it is determined that the virtual process is the substantially unique target of the call.

15. A computer program product for increasing the execution performance of a function at run-time, the computer program product comprising:

computer code that compiles the function;

computer code that identifies a call to a process, the call to the process being included in the function;

computer code that adds dependency information to the function, wherein the dependency information is arranged to indicate a status of the function, the status being arranged to include when a validity status of the function; and

a computer-readable medium that stores the computer codes.

16. A computer program product as recited in claim 15 wherein the process is a virtual process, the computer program product further including:

computer code that analyzes a class structure associated with the function, wherein analyzing the class structure includes determining when the virtual process is a substantially unique target of the call.

17. A computer program product as recited in claim 16 further including:

computer code that inlines the virtual process into the function when it is determined that the virtual process is the substantially unique target of the call.

18. A computer program product as recited in claim 16, the computer program product further including:

computer code that places a direct call to the virtual process in the function.

19. A computer program product as recited in claim 15 further including:

computer code that determines when the function is suitable for compilation.

20. A computer program product as recited in claim 15 further including:

computer code that loads a class, the class being associated with the function; and

computer code that determines when the dependency information indicates that the function is valid, that the function is suitable for deoptimization, and that the function is suitable for reoptimization.

21. A computer program product as recited in claim 15 wherein the computer-readable medium is one selected from the group consisting of a floppy disk, a hard disk, a tape, a data signal embodied in a carrier wave, a CD-ROM, a system memory, and a flash memory.

22. A compiled method structure for use in a computing environment, the compiled method structure including:

a compiled code portion, the compiled code portion being arranged to include an optimized call to at least one virtual function; and

a header portion, the header portion being arranged to include dependency information, the dependency information including information relating to a class associated with the at least one virtual function, information relating to a name of the at least one virtual function, and information relating to a signature of the at least one virtual function.